ELSEVIER

Science of Computer Programming 26 (1996) 99–115

Science of Computer Programming

Towards a design calculus for CSP

Rudolf Berghammer, Burghard von Karger *

Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Preusserstraße 1-9, D-24105 Kiel, Germany

Abstract

The algebra of relations has been very successful for reasoning about possibly non-deterministic programs, provided their behaviour can be fully characterized by just their initial and final states. We use a slight generalization, called *sequential algebra*, to extend the scope of relation-algebraic methods to reactive systems, where the behaviour between initiation and termination is also important. The sequential calculus is clearly differentiated from the relational one by the absence of a general converse operation. As a consequence, the past can be distinguished from the future, so that we can define the temporal operators and mix them freely with executable programs. We use a subset of CSP in this paper, but the sequential calculus can also be instantiated to different theories of programming. In several examples we demonstrate the use of the calculus for specification, derivation and verification.

1. Introduction

The main theme of this paper is the marriage of sequential and temporal reasoning in a single calculus. This combination is desirable because it allows mixing temporal connectives with imperative programming constructs. As a consequence, we may use equational reasoning for verifying a program or even calculating it from a specification. Our target is a subset of CSP, a process language for describing concurrent agents that cooperate via synchronous communication [12]. CSP is the conceptual core of occam, the language of choice for programming transputer networks.

Since most software errors result from erroneous descriptions of its intended behaviour, it is vital for specifications to be as clear and concise as possible. For this reason, we do not hesitate to employ logical and temporal operators in specifications even though they are not features of an implementable language like CSP. A *design calculus* which can bridge the gap between high-level descriptions of desired systems behaviour and executable code must combine elements of both into a single theory so

* Corresponding author.

SSDI 0167-6423(95)00024-0

that we can transform a specification gradually into a program through a series of local modifications each of which is justified by a law of the calculus. This calculational approach to program derivation has a strong tradition (see [6, 21, 4, 18]) and has been particularly successful in the realm of functional programming.

The denotational semantics of CSP [4, 22] and occam [11] has been used to establish algebraic identities between processes [14]. The collection of these laws constitutes a *program calculus*, which has been used for machine-supported transformations [10]. To upgrade the program calculus to a design calculus we need to extend the language and its semantics with specification constructs. Unfortunately, most interesting high-level constructs, especially the temporal connectives, are not definable in the standard models of CSP given in [4, 22]. We therefore address the challenge of constructing a semantical domain in which the operations of CSP and temporal logic can be modelled together.

Although we use CSP in this paper, we aim at a theory that does not depend on the choice of any particular programming language. In principle, our approach applies to all theories that describe systems as sets whose elements represent single observations of a single experiment. In this approach, non-deterministic choice is modelled as set union, refinement as set inclusion and, cum grano salis, parallel composition as set intersection. The set of all such system descriptions has the structure of a complete Boolean algebra. Besides the purely set-theoretic operations we also need to express time-wise composition of systems. The addition of an associative sequence operation leads from Boolean algebra to sequential algebra [15, 17], which is similar to the calculus of relations [1] but lacks a converse operation $R \mapsto R^{\mathsf{T}}$. This omission is justified by the irreversible nature of observations on reactive systems: Once an event (a communication) has happened, it cannot be cancelled or undone by what comes after. To make up for the lack of a general converse we introduce left and right cut operations which allow a style of reasoning very similar to that of the relational calculus.

By abandoning the converse operation we accept the irreversibility of time and enable temporal reasoning. In the absence of time-reversing observations, *all the temporal operations can be defined in terms of sequential composition.* This basic insight allows us to conceive linear temporal logic as a subtheory of the sequential calculus. In [16] we have shown that all the axioms of the complete proof system given in [19] are in fact theorems of the sequential calculus.

By carefully choosing the space of all possible observations we can instantiate the sequential calculus to a subset of CSP, which may then be enriched with additional operators and axioms. However, the construction of a design calculus for CSP is a very ambitious project, and a first approach must make some simplifying assumptions. The program calculus of CSP started from the trace model; only when that was well-understood it was extended to capture deadlock, livelock, timing, probability, etc. Similarly, a design calculus for CSP must start from a trace model; more sophisticated

[1] The axiomatic version of the relational calculus was developed by Tarski and his co-workers (see e.g. [26, 7]). Some applications to computer science can be found in [2, 23, 5, 13, 1].

observables may be added later. To retain extensibility we will endeavour to develop the theory as model-independent as possible.

In Section 2 we develop the algebra of sequential composition on which the sequential calculus is founded and use it as an algebraic foundation of temporal logic. In Section 3 we move to many-sorted algebra; this gives us operations for hiding, concurrency, and communication. The purpose of these two sections is to integrate both CSP and temporal operators in sequential algebra. The resulting calculus is a powerful specification tool which we illustrate with various examples in Section 4. There we also show how specifications may be used to calculate programs. In the concluding section, we indicate further applications of the ideas underlying this paper.

2. Boolean operations, sequence and cut

The hardest task in modelling reactive systems is to choose the right set of possible observations. If observations are input/output pairs then systems can be modelled as binary relations and we can use Tarski's relational calculus. When observations may be strings, the calculus of regular expressions applies. For real-time systems, functions from time intervals may be used to record observations, and various temporal logics have been designed for reasoning about them.

The precise definition of what constitutes an observation can be changed in a myriad ways and this has led to as many theories of programming. To reduce the amount of religious warfare, we suggest that a general theory of programming should not fix any particular observation space. More specific theories can still be obtained by restricting the universe of allowed observation spaces. Under this discipline theories can be arranged hierarchically and much duplication is avoided.

If we want a programming theory that is not only general but also useful we cannot allow arbitrary sets as observation spaces. Nothing can be achieved without some structure but whatever restrictions we impose will exclude some theories of interest. Fortunately, there is a considerable amount of common structure in many observation spaces, including the ones mentioned above. This is the subject of Sections 2.1 and 2.2 below.

Systems are represented as sets of observations and we obtain a calculus for systems by lifting the operations on single observations to the set level. We can then use the properties of observations to deduce laws for systems, but such reasoning in terms of individual observations does not deserve to be called a calculus. Instead we aim at an equational theory of systems that relies only on operations defined and laws valid for observation *sets*. To get started, we carefully select a small number of theorems and take them as *axioms* of sequential algebra. Direct reasoning is thereby confined to checking that the initial set of laws is indeed valid for every observation space, and all other theorems must be derived from the axioms. The body of derived theorems forms the sequential calculus which we introduce in Section 2.3.

2.1. Observation spaces

We are mainly interested in properties shared by many different kinds of observations. The most basic property is the existence of a *composition operation* which makes a possibly longer observation $x;y$ from sub-observations x and y. We require the associative law

$$(x;y);z = x;(y;z). \tag{O_1}$$

Composition need not be total. For example, if R_1 and R_2 are relations then the composition of $(r_1,s_1) \in R_1$ and $(r_2,s_2) \in R_2$ is only defined if $s_1 = r_2$, and when this equality holds then $(r_1,s_1);(r_2,s_2) =_{\text{def}} (r_1,s_2)$. To help reasoning about the definedness of composition, we introduce two functions between observations. Each observation x has a *left unit* $\overleftarrow{x}$ and a *right unit* $\overrightarrow{x}$, which satisfy the unit properties for composition:

$$\overleftarrow{x};x = x, \qquad x;\overrightarrow{x} = x. \tag{O_2}$$

For example, in the relational case the left unit of (r,s) is (r,r) and the right unit of the same pair is (s,s). Now definedness of composition is described by the law

$$x;y \text{ is defined} \iff \overrightarrow{x} = \overleftarrow{y}. \tag{O_3}$$

If $x = \overleftarrow{y}$ or $x = \overrightarrow{y}$ for some y then x is called a *unit*. The unit functions map units to themselves,

$$x \text{ is a unit} \iff \overleftarrow{x} = x = \overrightarrow{x}. \tag{O_4}$$

In the case where $x;y$ is defined, we also require that

$$\overleftarrow{x;y} = \overleftarrow{x}, \qquad \overrightarrow{x;y} = \overrightarrow{y}. \tag{O_5}$$

Two composite observations with identical first parts are equal only if they also have identical second parts (and vice versa). This is expressed by the rules of cancellation

$$x_1;y = x_2;y \implies x_1 = x_2, \qquad x;y_1 = x;y_2 \implies y_1 = y_2. \tag{O_6}$$

The purpose of this postulate is to ensure that the equation $a;x = b$ has at most one solution x. If a solution exists, it is denoted $a\text{-};b$. The left cut operator -; and its twin, the right cut operator ;- are characterized by

$$x;y = z \iff x = z;\text{-}y \iff y = x\text{-};z. \tag{O_7}$$

To summarize, an *observation space* $\mathcal{O}$ is a non-empty set with (partially defined) operations $x;y$, $x;\text{-}y$, and $x\text{-};y$ and unary functions $\overleftarrow{x}$ and $\overrightarrow{x}$ such that (O_1)–(O_7) hold[2].

[2] In other words: a small category in which every arrow is epic and monic.

2.2. Observations in temporal logic

In this section we suggest an interpretation of temporal formulae in terms of observations. Traditionally, the semantics of temporal logic is given as a satisfaction relation $\models$. For example,

$$\sigma, i \models p$$

signifies that the formula p holds at position i of the sequence σ. Satisfaction is defined recursively. A typical clause is

$$\sigma, i-1 \models \bigcirc p \quad \text{if and only if } i > 0 \text{ and } \sigma, i \models p.$$

This definition suggests regarding formulae as denotations for sets of "observations". Indeed it is convenient to identify every formula p with the set of all pairs (σ, i) such that $\sigma, i \models p$.

An observation (σ, i) refers to a single point in time. As a consequence, there is no natural way of composing two observations. Points cannot be composed, but intervals can. Moreover, points may be regarded as special intervals. Therefore, we move to a richer structure. For any set Σ of traces (finite or infinite sequences) let

$$\mathcal{O}_\Sigma =_{\text{def}} \{(\sigma, i, j) \mid \sigma \in \Sigma \text{ and } 0 \leqslant i \leqslant j \leqslant |\sigma|\}.$$

An observation (σ, i, j) may be pictured as a window with contents σ and currently visible part $\sigma_{i+1} \ldots \sigma_j$:

$$\underbrace{\sigma_0, \ldots, \sigma_i}_{\text{past}}, \overbrace{\sigma_{i+1}, \ldots, \sigma_j}^{\text{present}}, \underbrace{\sigma_{j+1}, \ldots}_{\text{future}}$$

Any of the three parts can be empty and we specifically allow i and j to take the value ∞ (infinity). An observation is a unit if its middle part is empty ($i = j$). Specifically, the left and right unit of (σ, i, j) are (σ, i, i) and (σ, j, j). Composition is defined by

$$(\sigma, i, j); (\sigma, j, k) = (\sigma, i, k).$$

We invite the reader to check that $\mathcal{O}_\Sigma$ satisfies the observation space postulates (O_1)–(O_7). But $\mathcal{O}_\Sigma$ has some additional structure which we shall now investigate.

The units of $\mathcal{O}_\Sigma$ cannot be decomposed. More precisely, we have

$$x; y \text{ is a unit} \implies x \text{ and } y \text{ are units}. \tag{O_8}$$

In other words, units are the only observations that have an inverse.

To formalize the idea that time can only progress in a single dimension, we introduce a prefix preorder on observations. We call x a *prefix* of y (and write $x \preceq y$) if there is some z with $x; z = y$. No observation can have two incomparable prefixes:

$$(x \preceq z) \wedge (y \preceq z) \implies (x \preceq y) \vee (y \preceq x). \tag{O_9}$$

Another distinguishing feature of $\mathcal{O}_\Sigma$ is that each observation carries within it a knowledge of the entire past and future. Two windows that have the same left or right unit must have the same contents and can differ only in size. In the case of left units this is expressed by

$$\overleftarrow{x} = \overleftarrow{y} \implies (x \preceq y) \vee (y \preceq x). \tag{O_{10}}$$

In other words: from each state there is essentially only one way to continue.

As expected, we can define temporal connectives on the observations in $\mathcal{O}_\Sigma$. For example, the "next" operator is defined by

$$\bigcirc(\sigma, i, j) =_{\text{def}} (\sigma, i-1, j-1) \quad \text{provided } i > 0.$$

If we identify the point observation (σ, i) with the interval observation (σ, i, i), then this definition of "next" corresponds to the earlier definition, in the sense that $\bigcirc p = \{\bigcirc x \mid x \in p\}$.

2.3. Sequential calculus

The objective of sequential algebra is to formalize a calculus of the subsets of an observation space $\mathcal{O}$. The powerset $2^{\mathcal{O}}$ obviously forms a complete Boolean algebra with union $\cup$, intersection $\cap$, complement $\overline{\ }$, ordering $\subseteq$, least element $\mathsf{O} =_{\text{def}} \emptyset$ and greatest element $\mathsf{U} =_{\text{def}} \mathcal{O}$. Just as relational composition is a lifted form of the composition of pairs (cf. Section 2.1), our more general sequential composition is obtained by lifting the composition defined for single observations to sets by

$$P; Q =_{\text{def}} \{z \mid \exists x \in P, y \in Q : x; y = z\}. \tag{S_1}$$

The identity element of composition is the set of all units:

$$\mathrm{I} =_{\text{def}} \{x \mid \overleftarrow{x} = x = \overrightarrow{x}\}. \tag{S_2}$$

The cutting operators are lifted in the same way. For example, each observation of $P \,;\!\text{-}\, Q$ is obtained from an observation of P by cutting from its right something that is an observation of Q:

$$P \,;\!\text{-}\, Q =_{\text{def}} \{z \mid \exists x \in P, y \in Q : x \,;\!\text{-}\, y = z\}. \tag{S_3}$$

Similarly for the left cut operator:

$$P \text{-}\!; Q =_{\text{def}} \{z \mid \exists x \in P, y \in Q : x \text{-}\!; y = z\}. \tag{S_4}$$

The cutting operators offer some compensation for the absence of a general converse because we can use $P \,;\!\text{-}\, Q$ to play the same role that $P; Q^{\mathsf{T}}$ plays in the relational calculus.

Citation from [9]: "We now take what is a standard step in mathematical theory building. The step is taken after the introduction of a mathematical novelty–such as

a new abbreviation or a mathematical macro–for formulae that were interpreted in a familiar domain of discourse. The step consists of starting with a clean slate and axiomatizing afresh the manipulations of the new formulae. In doing so, one creates a new domain of discourse; the role of the old, familiar domain of discourse, that used to constitute the subject matter, is thereby reduced to that of providing a possible model for the newly postulated theory. It is essential that the axioms of the new theory–which can be interpreted as theorems in the old universe of discourse–are clearly postulated as such and that the new theory is derived from them without reference to the model of the old universe of discourse. This is the only way of ensuring that the axioms of the new theory provide an interface that is independent of the old universe of discourse and that, hence, the new theory is safely applicable to alternative models."

Definition 2.1. A *sequential algebra* $\mathscr{S}$ is a complete Boolean algebra with three extra binary operations $P;Q$, $P\text{-};Q$ and $P;\text{-}Q$ such that $(\mathscr{S},;)$ is a monoid with identity I and

$$P;Q \subseteq \overline{R} \iff P\text{-};R \subseteq \overline{Q} \iff R;\text{-}Q \subseteq \overline{P} \quad \text{(Schröder axiom)},$$
$$P;(Q;\text{-}R) \subseteq (P;Q);\text{-}R \quad \text{(Euclidean axiom)},$$
$$P\text{-};I = I;\text{-}P \quad \text{(Reflection axiom)}.$$

The reasons behind our particular choice of axioms cannot be given here, but they are explained very carefully in [15]. In that paper we also prove that the Euclidean axiom implies its time-wise dual. As a consequence, the sequential calculus enjoys a perfect symmetry between past and future.

The Schröder axiom is familiar from the relational calculus (see [24]), except that $P;Q^{\mathsf{T}}$ and $P^{\mathsf{T}};Q$ are replaced by $P;\text{-}Q$ and $P\text{-};Q$. With this definition, the other axioms are also valid, so every relational algebra is a sequential algebra. The converse is false: There are many more sequential algebras than relational ones. In fact, with the definitions (S_1)–(S_4) the powerset over any observation space $\mathscr{O}$ is a sequential algebra, which we call the *sequential set algebra* over $\mathscr{O}$.

2.4. *Fixed point calculus*

The fixed point theorem of Knaster and Tarski guarantees that every monotonic function f on a sequential algebra has a least fixed point μf and a greatest fixed point νf. Iteration can be defined in terms of recursion, and we distinguish two types of loops. If P is an element of a sequential algebra then

$$P^{\omega} =_{\text{def}} \nu X \,.\, P;X$$

describes an infinite loop with body P. A loop that may also terminate after finitely many iterations (but not before the first one) is defined by

$$P^{+} =_{\text{def}} \nu X \,.\, P \cup P;X \,.$$

These definitions are somewhat questionable when P can terminate immediately. We avoid any problems by using them only for $P \subseteq \overline{I}$.

Fixed points are the subjects of a rich and elegant calculus [20]. We mention a few laws that we need in the examples. The first rule allows to unroll an infinite loop.

$$(P;Q)^\omega = P;(Q;P)^\omega . \tag{1}$$

In an infinite loop it does not matter if the body is repeated once or several times in each iteration:

$$(P^+)^\omega = P^\omega . \tag{2}$$

An infinite alternation of P and Q can be seen as an infinite loop that executes either P or Q at each step

$$(P;Q)^\omega \subseteq (P \cup Q)^\omega . \tag{3}$$

These rules can easily be proved within the calculus exposed in [20] (or directly, if you prefer).

2.5. *The temporal connectives*

If a (possibly partial) "next" operation is defined for observations then we can lift it to the set level by

$$\bigcirc P =_{\text{def}} \{\bigcirc x \mid x \in P\}. \tag{4}$$

The reason why we did not include any temporal connectives in the definition of a sequential algebra is that they can be *defined* in terms of the operators we already have. Before we come to this definition we need to discuss *transitions*.

The smallest measurements of progress are non-unit observations that cannot be further decomposed into non-unit sub-observations. Such observations correspond to single transitions of the observed system. The set of all transitions is formally described as

$$\mathsf{step} =_{\text{def}} \overline{I} \cap \overline{\overline{I}\,\overline{I}} .$$

Specifically, an observation $(\sigma, i, j) \in \mathcal{O}_\Sigma$ is a transition if $i + 1 = j < \infty$. Recall that we defined $\bigcirc(\sigma, i, j) = (\sigma, i - 1, j - 1)$, so we have $x = \bigcirc y$ just when there are transitions u and v with $x;u = v;y$. By (O_7) the latter equation is equivalent to $x = (v;y)\,;\text{-}\,u$. Thus we have proved, for every $P \subseteq \mathcal{O}_\Sigma$,

$$\bigcirc P = (\mathsf{step};P)\,;\text{-}\,\mathsf{step} . \tag{5}$$

In contrast to the original definition (4), no observations are mentioned here. Therefore, we take (5) as *defining* the "next" operator for an arbitrary sequential algebra. Its time-wise dual is the "previous" operator:

$$\ominus P =_{\text{def}} \mathsf{step}\,\text{-};(P;\mathsf{step}).$$

The other operators of temporal logic can be defined in terms of $\bigcirc$ and $\ominus$ and the least fixed point operator μ. In particular,

$$\begin{aligned}
\mathsf{first} &=_{\mathrm{def}} \overline{\ominus \mathsf{U}} & \mathsf{last} &=_{\mathrm{def}} \overline{\bigcirc \mathsf{U}} \\
p\ \mathit{Until}\ q &=_{\mathrm{def}} \mu x\ .\ q \cup (p \cap \bigcirc x) & p\ \mathit{Since}\ q &=_{\mathrm{def}} \mu x\ .\ q \cup (p \cap \ominus x) \\
\Diamond p &=_{\mathrm{def}} \mathsf{U}\ \mathit{Until}\ p & \boxminus p &=_{\mathrm{def}} \mathsf{U}\ \mathit{Since}\ p \\
\Box p &=_{\mathrm{def}} \overline{\Diamond \overline{p}} & \boxminus p &=_{\mathrm{def}} \overline{\Diamond \overline{p}}. \\
p\ \mathit{Unless}\ q &=_{\mathrm{def}} (p\ \mathit{Until}\ q) \cup \Box p & p\ \mathit{Backto}\ q &=_{\mathrm{def}} (p\ \mathit{Since}\ q) \cup \boxminus p .
\end{aligned}$$

To help the reader's intuition we repeat some of these definitions in set-theoretic form for the case where the underlying observation is $\mathcal{O}_\Sigma$:

$$\begin{aligned}
\mathsf{first} &= \{(t,i,j) \in \mathcal{O}_\Sigma \mid i = 0\}, \\
\Diamond p &= \{(t,i,j) \in \mathcal{O}_\Sigma \mid \exists k < \infty : (t, i+k, j+k) \in p\}, \\
\Box p &= \{(t,i,j) \in \mathcal{O}_\Sigma \mid \forall k < \infty : (t, i+k, j+k) \in p\}, \\
p\ \mathit{Until}\ q &= \{(t,i,j) \in \mathcal{O}_\Sigma \mid \exists k < \infty : (t, i+k, j+k) \in q \\
&\qquad \wedge \forall l < k : (t, i+l, j+l) \in p\}.
\end{aligned}$$

The quantifications over k are intended to range only over those values of k for which $(t, i+k, j+k)$ is a well-defined observation. That is, $j+k$ must be at most $|t|$ and when $j = \infty$ then k must be zero.

It is immediate from the Schröder axiom that $\bigcirc$ and $\ominus$ are each other's Galois conjugates:

$$\bigcirc P \subseteq \overline{Q} \iff \ominus Q \subseteq \overline{P}. \tag{6}$$

The extra properties of the observation spaces $\mathcal{O}_\Sigma$ allow us to strengthen the axioms of sequential algebra. Using (O_8) we can improve the Reflection axiom to

$$I \,;\text{-}\, \mathsf{U} = I . \tag{7}$$

Thanks to (O_9) we may replace the Euclidean axiom by the following equation:

$$(P;Q) \,;\text{-}\, R \;=\; P;(Q \,;\text{-}\, R) \;\cup\; P \,;\text{-}\, (R \,;\text{-}\, Q). \tag{8}$$

And, finally, (O_{10}) gives rise to the following new axiom:

$$(P \,;\text{-}\, Q);R \;\cup\; (Q \,;\text{-}\, P)\text{-};R \;=\; P \,;\text{-}\, (R\text{-};Q) \;\cup\; P;(Q\text{-};R). \tag{9}$$

From the stronger axioms one can show that

$$\bigcirc \ominus P \subseteq P, \qquad \ominus \bigcirc P \subseteq P . \tag{10}$$

Essentially all the laws of linear temporal logic follow from (6) and (10). We refer the interested reader to [16].

The **step** process is also very useful for modelling synchronous calculation. A typical law that can be proved within the sequential calculus is the following *lockstep rule*:

$$P^\omega \cap Q^\omega \;=\; (P \cap Q)^\omega \quad \text{provided } P, Q \subseteq \mathsf{step}. \tag{11}$$

3. Concurrency

In this section, we model concurrent processes in sequential algebra. Every CSP process is associated with a specific set of events it can engage in, called its alphabet; see [12]. In general, concurrent agents operate on different sets of events, whereas a single sequential algebra can only model processes with the same alphabet.

Let $\mathscr{E}$ be the universe of all possible events. For each subset A of $\mathscr{E}$ let $\mathscr{S}_A$ denote the sequential set algebra over $\mathscr{O}_{\Sigma(A)}$, where $\Sigma(A)$ is the set of all traces (finite and infinite sequences) over A. We call an element of a sequential algebra a *process*. For convenience, we shall pretend that $\mathscr{S}_A \cap \mathscr{S}_B = \emptyset$ for $A \neq B$ since this allows the *alphabet* $\alpha.P$ of a process P to be defined by

$$\alpha.P = A \iff P \in \mathscr{S}_A .$$

All operations defined so far operate on processes with a fixed alphabet. In contrast, parallel composition and hiding relate processes with different alphabets. In Section 3.1, we introduce the hiding operator as a new primitive that relates sequential algebras over different alphabets. Parallel composition and communication need not be introduced as new primitives because they can be *defined* in terms of hiding and its adjoint, the inserting operation.

3.1. Hiding and inserting

Let $A \subseteq B \subseteq \mathscr{E}$. On single observations, the hiding operation $\downarrow_A : \mathscr{O}_B \to \mathscr{O}_{B-A}$ is defined by

$$(t,i,j)\downarrow_A =_{\text{def}} (t\downarrow_A, i', j'),$$

where $t\downarrow_A$ results from t by deleting all occurrences of elements in A, i' is the number of indices $n \leqslant i$ with $t_n \notin A$, and, similarly, j' is the number of indices $n \leqslant j$ with $t_n \notin A$. Note that $t\downarrow_A$ may be finite even when t is infinite. Thus hiding can transform an infinite trace of visible events into a divergent run: An observation with a finite trace indicates a process that at some time engages into an infinite internal computation.

Like all functions defined on observations, hiding can be lifted to a set level operation $\downarrow_A : \mathscr{S}_B \to \mathscr{S}_{B-A}$, defined by

$$P\downarrow_A =_{\text{def}} \{x\downarrow_A \mid x \in P\}.$$

Since hiding is universally disjunctive (distributes over all unions), it has an upper adjoint $\uparrow^A : \mathscr{S}_{B-A} \to \mathscr{S}_B$, called *inserting*, which is uniquely characterized by the Galois connection

$$P \subseteq Q\uparrow^A \iff P\downarrow_A \subseteq Q. \tag{12}$$

Inserting can also be defined more directly by

$$Q\uparrow^A = \{x \in \mathscr{O}_B \mid x\downarrow_A \in Q\}.$$

We abbreviate $\downarrow_{\{a\}}$ and $\uparrow^{\{a\}}$ to $\downarrow_a$ and $\uparrow^a$, respectively.

Having introduced a new operator into an abstract calculus by explaining it for one of its concrete models, we have an obligation to give an axiomatization. Otherwise the advantages of abstractions will be lost and we will be reduced to conduct proofs in the concrete model. The following set of axioms is sufficient for the purposes of this paper, but we will not vouch for its completeness.

1. The Galois connection (12) between hiding and inserting.
2. $P\uparrow^A\downarrow_A = P$.
3. Inserting distributes over composition, quotients, and all unions.
4. Hiding is cumulative in the sense that $P\downarrow_A\downarrow_B = P\downarrow_{A\cup B}$.
5. Commutativity $P\downarrow_A\uparrow^B = P\uparrow^B\downarrow_A$ holds for disjoint sets A and B.
6. For the identity one has $I\downarrow_A = I$, $I_\emptyset\uparrow^A = \mathsf{U}_A$, and $I_A \parallel I_B = I_{A\cup B}$.

The last axiom mentions the parallel composition operator which is defined in the next section.

3.2. Parallel composition and communication

The parallel composition $P \parallel Q$ of two processes with the same alphabet is just their intersection. In other words, any given observation can be made on $P \parallel Q$ only when neither P nor Q prevents it. In the general case, the intersection can only be formed after lifting P and Q to their least common alphabet. We define

$$P \parallel Q =_{\text{def}} P\uparrow^A \cap Q\uparrow^B,$$

where $A = \alpha.Q - \alpha.P$ and $B = \alpha.P - \alpha.Q$. Thus, the two processes P and Q must synchronize on events that are in the intersection $\alpha.P \cap \alpha.Q$, but either can proceed independently with an event that belongs only to its own alphabet.

Let $a \in A$. We want to define a process $\mathsf{a} \in \mathscr{S}_A$ that performs a single a event and then terminates. We need an auxiliary definition. The process $only(a)$ can perform an arbitrary (even infinite) number of a events. Let I denote the identity of $\mathscr{S}_{\mathscr{A}}$. We define

$$\begin{aligned} only(a) &= I\downarrow_a\uparrow^a \\ &= \{(t,i,j) \in \mathscr{S}_A \mid t_{i+1} = \cdots = t_j = a\}. \end{aligned}$$

Strictly speaking, we should write $only(a)_A$ instead of $only(a)$ but we will not burden ourselves with this heavy notation, since the alphabet can usually be reconstructed from the context. Now a process that will produce exactly one a event is given by

$$\mathsf{a} =_{\text{def}} \mathsf{step} \cap only(a).$$

When $a \in \alpha.P$ we may *prefix* P with the *communication* a. The resulting process $\mathsf{a};P$ (which is written $a \to P$ in standard CSP) starts with the event a and then behaves like P.

4. Examples

This section is devoted to some examples that illustrate the power of the calculus to express specifications and its use in program derivation and verification.

4.1. Specifications

Traditionally, descriptions of CSP processes yet to be written have been set down directly in the semantic domain using full set theory rather than the restricted notation of a calculus [12, 22, 8]. Such specifications often have the advantage of simplicity and clarity, but they cannot be transformed into programs by algebraic calculation and they are hard to mechanize. The purpose of this section is to show that specifications can be written inside the many-sorted sequential calculus.

Oscillator 4.1. An oscillator has two states. In one state it always sends a, and in the other it always sends b. It cannot stay in one state forever. A temporal specification can be given by the alphabet $\alpha.\mathit{oscillator} =_{\mathrm{def}} \{a, b\}$ and the inclusion

$$\mathit{oscillator} \subseteq (\Diamond \mathtt{a} \cap \Diamond \mathtt{b})^{\omega}.$$

In words, it is true at each step that there exists both a future `a` and a future `b` step. Such liveness conditions cannot be stated in standard CSP.

Fire alarm 4.2. Once a fire signal has been received (e.g., from a sensor), a fire alarm has to ring the bell continuously, unless someone presses the reset button. Its alphabet consists of the events *fire*, *bell*, and *reset* and we require

$$\mathit{fire\text{-}alarm} \subseteq (\mathtt{fire} \rightarrow \bigcirc(\mathtt{bell}\ \mathit{Unless}\ \mathtt{reset}))^{\omega}.$$

It would be erroneous to use *Until* in place of *Unless*, because a fire alarm cannot enforce that the reset button is pressed eventually. Another clause is added to the specification to forbid false alarms:

$$\mathit{fire\text{-}alarm} \subseteq (\mathtt{bell} \rightarrow \Diamond\!\!\!-\mathtt{fire})^{\omega}.$$

If we also wish to enforce that the reset button always works we just replace $\Diamond\!\!\!-\mathtt{fire}$ by the stronger requirement $\overline{\mathtt{reset}}\ \mathit{Since}\ \mathtt{fire}$.

Buffer 4.3. This example is taken from [22]. The alphabet of a buffer contains (an input channel) a and (an output channel) b. Whenever an input has been received, no further input is allowed before the next output, and it can only output if it has not done so since the last input. This behaviour is captured in the following specification:

$$\mathit{buffer} \subseteq ((\mathtt{a} \rightarrow (\overline{\mathtt{a}}\ \mathit{Unless}\ \mathtt{b})) \cap (\mathtt{b} \rightarrow (\overline{\mathtt{b}}\ \mathit{Since}\ \mathtt{a}))^{\omega}.$$

From a practical point of view, this specification is silly. It is much harder to understand than the natural implementation of a buffer, and it does not generalize to buffers of

capacity greater than one. A much more desirable specification of an n-place-buffer (n-buffer for short) just states that at any time, the number of inputs received so far must at least equal, but not exceed by more than n, the number of outputs produced so far:

$$buffer(n) \subseteq (0 \leqslant \#a - \#b \leqslant n)^{\omega}.$$

The term $(0 \leqslant \#a - \#b \leqslant n)$ is intended to describe the set of all observations $(t, i, i+1)$ with $i < |t|$ such that the bag $\langle t_1, \ldots, t_{i+1} \rangle$ contains at least as many a's as b's, but at most n more a's than b's. It requires a little specification engineering to express this requirement in the calculus. First of all we note that we need only consider one-point ranges, because

$$(0 \leqslant \#a - \#b \leqslant n) =_{\text{def}} \bigcup_{0 \leqslant k \leqslant n} (\#a - \#b = k).$$

If we can count occurrences of a and b, we can also express the difference as

$$(\#a - \#b = k) =_{\text{def}} \bigcup_{r \geqslant 0} (\#a = r + k) \cap (\#b = r).$$

Next, a specification $\#a = k$ can be rewritten in terms of inequations:

$$(\#a = k) \;=_{\text{def}}\; (\#a \geqslant k) \cap \overline{\#a \geqslant k+1}.$$

Finally, the meaning of an inequation can be defined inductively by

$$(\#a \geqslant 0) \;=_{\text{def}}\; \mathsf{step}, \qquad (\#a \geqslant k+1) \;=_{\text{def}}\; \diamondminus(\mathsf{a} \cap \ominus(\#a \geqslant k)).$$

With these definitions we can use the calculus to establish algebraic laws for the counting operations. In particular, we have for distinct events a, b, and c:

$$\mathsf{a} \wedge \ominus(\#a - \#b = n) \;\subseteq\; (\#a - \#b = n+1), \tag{13}$$

$$\mathsf{first} \cap \mathsf{a} \;\subseteq\; (\#a - \#b = 1), \tag{14}$$

$$(\#a - \#b = n)\downarrow_B \;=\; (\#a - \#b = n) \quad \text{if } a, b \notin B, \tag{15}$$

$$(\#a - \#b = n)\uparrow^B \;=\; (\#a - \#b = n)^{+} \quad \text{if } a, b \notin B, \tag{16}$$

$$(\#a - \#b = n) \wedge (\#b - \#c = m) \;\subseteq\; (\#a - \#c = n + m). \tag{17}$$

The proofs of (13–17) are left to the reader.

4.2. Deriving programs

Next, we demonstrate how non-algorithmic problem specifications may be transformed into programs. We consider again buffers. Compared with the approach taken in [22], the new feature of our treatment is the free mixing of specification constructs

with implementable operations and the formal program derivation within a single algebraic framework.

Composing buffers 4.4. Let P be an n-buffer and Q an m-buffer with alphabets $\alpha.P = \{a, b\}$ and $\alpha.Q = \{b, c\}$ so that we have

$$P \subseteq (0 \leqslant \#a - \#b \leqslant n)^{\omega}, \qquad Q \subseteq (0 \leqslant \#b - \#c \leqslant m)^{\omega}.$$

We want to show that an $(n+m)$-buffer with input a and output c can be implemented by composing P and Q in parallel and hiding the connecting wire b:

$$\begin{aligned}
& (P \parallel Q)\downarrow_b \\
\subseteq\ & \{\text{assumptions on } P \text{ and } Q, \text{ definition of } \parallel\} \\
& ((0 \leqslant \#a - \#b \leqslant n)^{\omega}\uparrow^c \cap (0 \leqslant \#b - \#c \leqslant m)^{\omega}\uparrow^a)\downarrow_b \\
=\ & \{\text{Inserting distributes over loops}\} \\
& (((0 \leqslant \#a - \#b \leqslant n)\uparrow^c)^{\omega} \cap ((0 \leqslant \#b - \#c \leqslant m)\uparrow^a)^{\omega})\downarrow_b \\
=\ & \{(16), \text{ loop absorption rule } (2)\} \\
& ((0 \leqslant \#a - \#b \leqslant n)^{\omega} \cap (0 \leqslant \#b - \#c \leqslant m)^{\omega})\downarrow_b \\
=\ & \{\text{Lockstep rule } (11)\} \\
& ((0 \leqslant \#a - \#b \leqslant n) \cap (0 \leqslant \#b - \#c \leqslant m))^{\omega}\downarrow_b \\
\subseteq\ & \{(17)\} \\
& (0 \leqslant \#a - \#c \leqslant n + m)^{\omega}\downarrow_b \\
=\ & \{\text{Hiding distributes over loops, } (15)\} \\
& (0 \leqslant \#a - \#c \leqslant n + m)^{\omega}.
\end{aligned}$$

Implementing buffers 4.5. In the previous example we showed that arbitrary large buffers can be built from 1-buffers. Now we derive an implementation of a 1-buffer with input a and output b. For convenience we use the abbreviation $\Delta_n =_{\text{def}} (\#a - \#b = n)$.

$$\begin{aligned}
& \mathit{buffer} \\
=\ & \{\text{Definition of a buffer}\} \\
& (\Delta_0 \cup \Delta_1)^{\omega} \\
\subseteq\ & \{\text{Loop decomposition rule } (3)\} \\
& (\Delta_1; \Delta_0)^{\omega} \\
=\ & \{\text{Loop rolling rule } (1)\} \\
& \Delta_1; (\Delta_0; \Delta_1)^{\omega} \\
=\ & \{P; Q = P; (\ominus P \Rightarrow Q) \text{ provided } P, Q \subseteq \mathbf{step}\}
\end{aligned}$$

$$
\begin{array}{ll}
 & \Delta_1;((\ominus\Delta_1 \Rightarrow \Delta_0);(\ominus\Delta_0 \Rightarrow \Delta_1))^{\omega} \\
\subseteq & \quad \{(13)\} \\
 & \Delta_1;(b;a)^{\omega} \\
\subseteq & \quad \{(14)\} \\
 & (\mathsf{first} \cap a);(b;a)^{\omega} \\
= & \quad \{\text{Sequential calculus}\} \\
 & \mathsf{first} \cap a;(b;a)^{\omega} \\
= & \quad \{\text{Loop rolling rule (1)}\} \\
 & \mathsf{first} \cap (a;b)^{\omega}.
\end{array}
$$

Thus a 1-buffer can be implemented as an infinite loop. The conjunct **first** makes sure that the buffer starts up together with or before its clients. The derivation shows why it is necessary to place this obligation on the user of a buffer. Without it the buffer's invariant might be destroyed before it even starts.

5. Concluding remarks

We have exploited two new ideas in this paper. Firstly, temporal operations may be reduced to composition and quotients. This allowed us to integrate temporal logic into an imperative framework. Secondly, we moved from a single-sorted sequential algebra to a many-sorted family and appropriate hiding and inserting operations, in order to define the basic operations of CSP in an algebraic and model-independent way.

Both ideas are much more general than the scope in which they were applied here. In a companion paper [16] it is shown that it is possible to obtain various forms of temporal logic from sequential algebra by selecting a few additional axioms from a pool of alternatives. In particular, linear temporal logic and its complete proof system [19] have been worked out entirely within the calculus.

Hoare and Lamport have argued that parallel composition means intersection. Others believe it should be modelled as direct product. We believe that both parties are right – some of the time. $P \parallel Q = P \cap Q$ holds when P and Q have the same alphabet and $P \parallel Q$ may be seen as a direct product when the alphabets are disjoint. Using many sorts and the inserting operations, we covered both extremes, and all positions in-between. This technique applies not just to the sequential calculus. In the predicate calculus, hiding is just existential quantification. In fact, our definition of parallel composition has already been used in that context. It is precisely the conjunction of schemas in the Z specification language; see [25]. Unlike in the usual notation, the adjoint of existential quantification (inserting) can be expressed, which is very useful in the calculational style of reasoning. Other obvious candidates are the relational calculus, interval temporal logic, and Dijkstra's regularity calculus [9], all of which may be enriched with hiding, inserting, and parallel composition.

Acknowledgements

We are indebted to Tony Hoare who encouraged the emergence of sequential algebra, and suggested its application to CSP. Furthermore, we thank Jules Desharnais, Bernhard Möller, Martin Russling and Walter Vogler for valuable remarks.

References

[1] R.C. Backhouse, P. de Bruin, G. Malcolm, T.S. Voermans and J. van der Woude, Relational catamorphisms, in: B. Möller, ed., *Constructing programs from specifications, Proc. IFIP TC2 / WG2.1 Working Conf. on Constructing Programs* (Elsevier, Amsterdam, 1991) 287–318.

[2] J.W. de Bakker and W.P. de Roever, A calculus for recursive program schemes, in: M. Nivat, ed., *Proc. ICALP 73* (North-Holland, Amsterdam, 1973) 167–196.

[3] F.L. Bauer, B. Möller, H. Partsch and P. Pepper, Formal program construction by transformations – Computer-aided intuition guided programming, *IEEE Software Eng.* **15** (1989) 165–180.

[4] S.D. Brookes and A.W. Roscoe, An improved failure model for communicating sequential processes, in: S.D. Brookes et al., eds., *Proc. NSF-SERC Seminar on Concurrency*, Lecture Notes in Computer Science, Vol. 197 (Springer, Berlin, 1985) 285–305.

[5] R. Berghammer and H. Zierer, Relational algebraic semantics of deterministic and nondeterministic programs, *Theoret. Comput. Sci.* **43** (1986) 123–147.

[6] R.M. Burstall and J. Darlington, A transformation system for developing recursive programs, *J. ACM* **24** (1977) 44–67.

[7] L.H. Chin and A. Tarski, Distributive and modular laws in the arithmetic of relation algebras, Univ. of California Publications in Mathematics (new series) **1** (1951) 341–384.

[8] J. Davies, Specification and proof in real time systems. D. Phil. Thesis, Oxford Univ., 1991.

[9] E.W. Dijkstra, The unification of three calculi, in: M. Broy, ed., *Program Design Calculi* (Springer, Berlin, 1993) 197–231.

[10] M.H. Goldsmith, *The Oxford Occam Transformation System* (draft user documentation) (Oxford Univ. press, Oxford, 1994)

[11] M.H. Goldsmith, A.W. Roscoe and B.G.O. Scott, Denotational semantics for occam 2. Parts I and II, *Transputer Commun.* **1**(2) (1993) 65–91 and **2**(1) (1994) 25–67.

[12] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[13] C.A.R. Hoare and H. Jifeng, The weakest prespecification. Parts I and II, *Fundam. Inform.* **IX** (1986) 51–84 and 217–252.

[14] C.A.R. Hoare and A.W. Roscoe, The laws of occam programming, Technical Report PRG-53, Oxford Univ., 1986.

[15] B. von Karger, Sequential calculus, ProCoS II Report Kiel BvK 15/4, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, 1994.

[16] B. von Karger, An algebraic foundation of temporal logic, in: P.D. Mosses et al. eds., *Proc. TAPSOFT '95*, Lecture Notes in Computer Science, Vol. 915 (Springer, Berlin, 1995) 232–246. Also available as: ProCoS II Report Kiel BvK 17/1, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, 1994.

[17] B. von Karger and C.A.R. Hoare, Sequential calculus, *Inform. Process. Lett.* **53** (1995) 123–130.

[18] L. Lamport and S. Merz, Specifying and verifying fault-tolerant systems, in: H. Langmaack et al. eds., *Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science, Vol., 863 (Springer, Berlin, 1994) 41–76.

[19] Z. Manna and A. Pnueli, The temporal logic of reactive and concurrent systems-specification (Springer, Berlin, 1991).

[20] Mathematics of Program Construction Group Eindhoven, Fixed-point calculus, *Inform. Proces. Lett.* **53** (1995) 131–136.

[21] L.G.L.T. Meertens, Algorithmics – Towards programming as a mathematically activity, in: J.W. de Bakker et al., eds., *Proc. CWI Symp. on Mathematics and Computer Science*, CWI Monographs, Vol. 1 (North-Holland, Amsterdam, 1986) 289–334.

[22] E.R. Olderog and C.A.R. Hoare, Specification-oriented semantics for communicating processes, *Acta Inform.* **33** (1986) 9–66.
[23] G. Schmidt, Programs as partial graphs. Parts I and II, *Theoret. Comput. Sci.* **15** (1981) 1–25 and 159–179.
[24] G. Schmidt and T. Ströhlein, Relations and graphs, *Discrete Mathematics for Computer Scientists*, EATCS Monographs on Theoretical Computer Science (Springer, Berlin, 1993).
[25] J.M. Spivey, The *Z* notation: a reference manual (Prentice-Hall, Englewood Cliffs, NJ, 2nd ed., 1992).
[26] A. Tarski, On the calculus of relations, *J. Symbolic Logic* **6** (1941) 73–89.
[27] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, *Pacific J. Math.* **5** (1955) 285–309.